THE SOFTIES PRESENT:

TUTOR

ASSEMBLY LANGUAGE TUTORIAL FOR THE TEXAS INSTRUMENTS HOME COMPUTER

THE TEXAS INSTRUMENTS HOME COMPUTER
AND MINI-MEMORY MODULE ARE REQUIRED.

ADDENDUM
Tutor Assembley Tutorial

Please mark the following changes in your Tutor

Page	Lesson	Description
24	VII	Change DATA >02D2,SC,>0005 to DATA >02D2 DATA SC DATA >0005
24	VII	Change DATA >02EF,HS,>0008 to DATA >02EF DATA HS DATA >0008
25	VII	Change DATA >0284,OV,>0016 to DATA >0284 DATA OV DATA >0016

These same changes should be made to the game listing. These changes are syntax changes and will not change the assembled values.

TABLE OF CONTENTS

ASSEMBLY LANGUAGE TUTORIAL   Page

FORWARD....................................1

PRE-LESSON.................................2

LESSON I...................................8

LESSON II..................................10

LESSON III.................................12

LESSON IV..................................15

LESSON V...................................18

LESSON VI..................................21

LESSON VII.................................24

APPENDIX I...HEX TO DECIMAL

APPENDIX II..TV SCREEN

APPENDIX III.ASCII CODES

APPENDIX IV..GAME LISTING

TRUTH TABLE

INSTRUCTION TABLE

REFERANCE CARD

GLOSSARY

EVALUATION CARD

This manual was created for the Softies by Steve Barstad.

Additional Contributions: E.D.Barstad and J.Shima.

FORWARD

TUTOR IS DESIGNED TO AID YOU IN UNDERSTANDING ASSEMBLY LANGUAGE FOR THE TI99/4A. THE TOOLS NECESSARY TO INTERACT TUTOR WITH YOUR TI99/4A ARE:

1. MINI-MEMORY MODULE (MINIMEM)
2. CASSETTE TAPE PLAYER TO LOAD PROGRAMS NEW/LINES.
3. SOME BLANK CASSETTE TAPES

TO MAXIMIZE LEARNING IT IS RECOMMENDED THAT YOU ALSO PURCHASE THE EDITOR/ASSEMBLER OWNER'S MANUAL. THIS IS AVAILABLE FROM TEXAS INSTRUMENTS INCORPORATED, DALLAS, TEXAS OR THE **SOFTIES**, 7300 GALLAGHER #229, EDINA, MINNESOTA.

TUTOR IS THE FIRST IN A SERIES OF HELPFUL STEP BY STEP TEACHING AIDS FOR LEARNING ASSEMBLY LANGUAGE. TO GET THE MOST OUT OF TUTOR, START WITH THE PRE-LESSON AND CONTINUE UNTIL ALL THE LESSONS HAVE BEEN COMPLETED. MAKE SURE YOU FOLLOW ALL THE THE DIRECTIONS AND PERFORM THE SIMPLE EXERCISES THAT ACCOMPANY EACH LESSON. IF YOU ARE UNCERTAIN ABOUT SOMETHING GO BACK AND RE-READ THAT SECTION.

WHEN YOU ARE FINISHED, YOU WILL HAVE TYPED IN A SIMPLE GAME THAT RUNS IN ASSEMBLY LANGUAGE.

PRE-LESSON

IMAGINE THAT YOU ARE A FOREIGN DIPLOMAT AND YOU HAVE AN IMPORTANT MEETING WITH THE AMBASSADOR OF ANOTHER COUNTRY. IN ORDER TO COMMUNICATE WITH THE AMBASSADOR YOU MUST SPEAK THROUGH AN INTERPRETER. THIS CAN BE VERY VERY SLOW. THIS IS EXACTLY WHAT HAPPENS WHEN WE USE BASIC. WHEN WE RUN A BASIC PROGRAM, THE COMMANDS THAT WE WROTE ARE CONVERTED INTO MACHINE LANGUAGE INSTRUCTIONS BY THE BASIC INTERPRETER. WHAT TUTOR WILL ATTEMPT TO DO IS TO ELIMINATE THE MIDDLE MAN AND GIVE YOU A REMARKABLE SPEED INCREASE. TUTOR WILL TRY TO TEACH YOU TO COMMUNICATE WITH THE COMPUTER ON ITS OWN LEVEL.

YOUR TI UNDERSTANDS TWO NUMBER SYSTEMS IN THE MACHINE LANGUAGE MODE, THEY ARE CALLED **BINARY** AND **HEXADECIMAL**. NEITHER SYSTEM IS DIFFICULT TO LEARN ONCE YOU UNDERSTAND THE BASIC PRINCIPLES. YOU DO NOT HAVE TO BE A MATHEMATICAL GENIUS TO USE THEM. RELAX, TAKE A DEEP BREATH, AND READ ON.

LET'S BEGIN OUR DISCUSSION OF NUMBER SYSTEMS BY TAKING A LOOK AT THE NUMBER SYSTEM WE USE EVERYDAY. FROM THERE, IT IS EASY TO SEE THE SIMILARITIES BETWEEN THE SYSTEMS. THE NUMBER SYSTEM WE COMMONLY USE IS CALLED THE DECIMAL OR BASE TEN SYSTEM. IT COMES FROM THE LATIN ROOT DECIM MEANING TEN. WE DEVELOPED THE SYSTEM BECAUSE WE WERE BLESSED WITH TEN FINGERS, WHO KNOWS WHAT WOULD HAVE RESULTED IF WE WERE BLESSED WITH THIRTY-SEVEN FINGERS.

THE DECIMAL SYSTEM IS SET UP ON A WORKING BASE OF TEN. THIS NUMBER GIVES YOU TWO VERY IMPORTANT PIECES OF INFORMATION. FIRST, IT TELLS YOU HOW MANY DIFFERENT SYMBOLS ARE AVAILABLE FOR USE. (SINCE WE ARE DISCUSSING THE DECIMAL SYSTEM, WHERE THE BASE IS TEN, WE USE THE TEN SYMBOLS 0,1,2,3,4,5,6,7,8,9.) SECOND, THE BASE NUMBER TELLS US HOW TO ACTUALLY READ A NUMBER WRITTEN IN THE DECIMAL SYSTEM.

EXAMPLE:

LET'S LOOK AT THE NUMBER 1839, AND BREAK IT INTO ITS COMPONENT PARTS.

1	8	3	9
1000	100	10	1

THIS SAYS THAT THERE ARE:

9 ONES IN THE 1ST POSITION= 9
PLUS 3 * 10^1 IN THE 2ND POSITION= 30
PLUS 8 * 10^2 IN THE 3RD POSITION= 800
PLUS 1 * 10^3 IN THE 4TH POSITION=1000
1839

OR (1 * 1000) + (8 * 100) + (3 * 10) + (9 * 1) = 1839

BOTH BINARY AND HEXADECIMAL ARE SET UP ON EXACTLY THE SAME PRINCIPLES. THE MAIN DIFFERENCES ARE THE BASE NUMBER, THE AVAILABLE SYMBOLS AND THE POSITIONAL VALUE OF THE SYMBOLS. LET'S ATTACK BINARY FIRST.

BINARY COMES FROM THE LATIN ROOT BI MEANING TWO. IT HAS A WORKING BASE OF TWO. WE KNOW FROM OUR PREVIOUS DISCUSSION OF THE DECIMAL SYSTEM THAT BINARY ONLY GIVES US TWO WORKING SYMBOLS, NAMELY 0 AND 1. THE PLACE VALUES IN BINARY INCREASE BY POWERS OF TWO.

LET'S LOOK AT A BINARY NUMBER AND SEE IF WE CAN INTERPRET IT.

1	1	0	1
8	4	2	1

THIS WOULD BE:

```
  1 * 1 = 1
+ 0 * 2 = 0
+ 1 * 4 = 4
+ 1 * 8 = 8
         ---
         13
```

OR (1 * 8) + (1 * 4) + (0 * 2) + (1 * 1) = 13. THEREFORE THE DECIMAL EQUIVALENT OF THE BINARY NUMBER 1011 IS 13.

OKAY, SO ITS EASY TO INTERPRET A BINARY NUMBER INTO A DECIMAL NUMBER, BUT HOW DO YOU GET FROM A DECIMAL NUMBER TO A BINARY NUMBER. THE EASIEST WAY TO DO THIS IS TO PERFORM A SERIES OF DIVISIONS. FIRST LET'S SET UP THE FIRST FOUR PLACES IN THE BINARY SYSTEM.

	8	4	2	1	
1. CHOOSE A DECIMAL NUMBER BETWEEN 0 AND 15. WE'LL USE 9.				9	
2. START WITH THE HIGHEST PLACE VALUE. THAT VALUE IS 8. DIVIDE THE NUMBER BY THIS VALUE GIVING "1" AND A REMAINDER OF "1"				9/8 =	1 R 1
3. TAKE THE REMAINDER AND DIVIDE BY THE NEXT HIGHEST PLACE VALUE.				1/4 =	0 R 1
4. CONTINUE ON DIVIDING BY EACH SUBSEQUENT PLACE VALUE UNTIL ALL PLACES ARE FILLED.				1/2 = 1/1 =	0 R 1 1 R 0
5. NOW WE PLACE THE NUMBERS IN THEIR CORRECT POSITION AND WE ARE FINISHED.				1 0 0 1	

THIS MAY SEEM TEDIOUS SO HERE IS A BASIC PROGRAM:

```
10 INPUT A
20 IF A>15 THEN 10
30 IF A<0  THEN 110
40 FOR I = 3 TO 0 STEP -1
50 V = 2 ^ I
60 A1 = INT(A / V)
70 PRINT A1;" ";
80 A = A - A1 * V
90 NEXT I
100 PRINT
110 GOTO 10
120 STOP
```

NOW WE ARE READY FOR HEXADECIMAL. HEXADECIMAL COMES FROM THE GREEK WORD HEX MEANING SIX AND THE LATIN WORD DECIM MEANING TEN. THE COMBINATION OF THE TWO MEANS SIXTEEN. HEXADECIMAL IS A BASE SIXTEEN SYSTEM. THE PLACE VALUES IN HEXADECIMAL INCREASE BY POWERS OF SIXTEEN. WE KNOW THAT THERE ARE SIXTEEN WORKING SYMBOLS IN HEXADECIMAL BECAUSE THE BASE NUMBER TELLS US THIS. HOWEVER, THEY DO NOT FOLLOW THE STANDARD SYMBOL PATTERN OF 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 INSTEAD THE WORKING SYMBOLS OF HEXADECIMAL ARE 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. THE LETTERS TAKE THE PLACE OF THE TWO DIGIT NUMBERS, AS SUCH A=10, B=11, C=12, D=13, E=14, F=15. OTHER THAN THE UNIQUE SYMBOL PATTERN, HEXADECIMAL WORKS THE SAME AS BINARY AND DECIMAL. IN ALL REALITY HEXADECIMAL IS A SHORTHAND VERSION OF BINARY. IT SIMPLY CONDENSES FOUR BINARY PLACES INTO ONE HEXADECIMAL PLACE.

NOW LET'S TRY TO INTERPRET A HEXADECIMAL NUMBER. OUR NUMBER WILL BE:

1	3	A	E
4096	256	16	1

WE FIND THAT

$$
\begin{aligned}
14 * 16^0 &= 14 \\
10 * 16^1 &= 160 \\
3 * 16^2 &= 768 \\
1 * 16^3 &= \underline{4096} \\
& 5038
\end{aligned}
$$

OR (1 * 4096) + (3 * 256) + (10 * 16) + (14 * 1) = 5038

TO CHANGE A DECIMAL NUMBER TO HEXADECIMAL YOU MUST CONDUCT A SERIES OF DIVISIONS.

1.	SET UP FOUR HEX PLACES.	____ ____ ____ ____ 4096 256 16 1
2.	CHOOSE A DECIMAL NUMBER BETWEEN 0 AND 65535. WE WILL USE 1389.	1389.
3.	DIVIDE BY THE VALUE IN THE LEFTMOST PLACE	1389/4096 = 0 R 1389
4.	NOW DIVIDE BY THE NEXT HIGHEST HEX PLACE.	1389/256 = 5 R 109
5.	REPEAT THE PROCESS.	109/16 = 6 R 13 13/1 = D R 0
6.	NOW PLACE THE SYMBOLS IN THEIR CORRECT ORDER.	>056D

THIS PROCESS IS TIME CONSUMING AND THE DIVISION CAN GET MESSY, SO TO MAKE IT EASIER ON YOU TYPE IN THIS SIMPLE PROGRAM. THIS PROGRAM WILL CHANGE DECIMAL NUMBERS BETWEEN 0 AND 255 INTO HEXADECIMAL NUMBERS.

```
5  H$="0123456789ABCDEF"
10 INPUT A
20 IF A < 0 THEN 90
30 IF A > 255 THEN 10
40 T1 = INT(A/16)
50 T2 = A - (16 * T1)
60 PRINT SEG$(H$,T1+1,1);
70 PRINT SEG$(H$,T2+1,1)
80 GOTO 10
90 STOP
```

ANOTHER WAY TO CONVERT BETWEEN SYSTEMS IS TO USE TABLES (SEE APPENDIX ONE).

A FEW DEFINITIONS:

BIT IS AN ABREVIATION FOR BINARY DIGIT. A BIT CAN HAVE A VALUE EITHER 1 OR 0.

A NIBBLE IS A HEXADECIMAL DIGIT. IT IS AN ABREVIATION FOR FOUR BITS. A NIBBLE CAN HAVE A VALUE FROM >0 TO >F.

A BYTE IS TWO NIBBLES. >D4 IS A BYTE. THE LARGEST BYTE IS >FF.

A WORD IS TWO BYTES. IT IS ALSO FOUR NIBBLES, OR SIXTEEN BITS. >8375 IS A WORD.

GET USED TO SEEING THE ">" IN FRONT OF NUMBERS. IT WILL INDICATE THAT THE NUMBER IS A HEXADECIMAL NUMBER. IN THE LESSONS THAT FOLLOW, YOU WILL BE SEEING IT OFTEN.

ONE MORE THING AND WE WILL BE READY TO GO. CAREFULLY READ PAGES 4-6 OF THE <u>LINE-BY-LINE ASSEMBLER</u> MANUAL. FOLLOW THE INSTRUCTIONS TO INITIALIZE AND LOAD "LINES/NEW" INTO THE MODULE

NOW WE ARE READY TO GO. TAKE A DEEP BREATH, HOLD ON TO YOUR HAT, AND LETS BEGIN.

LESSON I

WELCOME TO THE WONDERFUL WORLD OF TI99 MACHINE LANGUAGE. WE HOPE THAT WHEN YOU ARE DONE WITH THIS TUTORIAL YOU WILL HAVE THE NECESSARY VOCABULARY AND WORKING KNOWLEDGE TO BE ABLE TO WRITE AND ENJOY MACHINE LANGUAGE.

THE TMS9900 IS A 16 BIT MACHINE. WHAT THIS MEANS IS THAT THE LENGTH OF MOST OF IT'S INSTRUCTIONS ARE 16 BITS (ONE WORD) LONG, IT TAKES 16 BITS TO UNIQUELY IDENTIFY ANY GIVEN MEMORY LOCATION, AND THE REGISTERS ARE 16 BITS LONG.

ALL COMPUTERS HAVE WHAT ARE CALLED REGISTERS. EACH COMPUTER USES AND IMPLEMENTS REGISTERS IN ITS OWN WAY. IN SOME MACHINES REGISTERS ARE USED VERY LITTLE. IN THE TI, THEY ARE USED ALOT!!!!! THEREFORE THE BEST PLACE TO START IS TO GIVE A QUICK DISCUSSION OF THE TI REGISTER. TI REFERS TO ITS REGISTERS AS WORKSPACE REGISTERS. THE REASON FOR THIS WILL BE EXPLAINED A LITTLE LATER. THERE ARE 16 OF THESE REGISTERS, EACH 16 BITS LONG. INTO ANY OF THESE REGISTERS CAN BE PUT 16 BITS OF INFORMATION. THE INFORMATION COULD BE DATA OR IT COULD BE AN ADDRESS. REGISTERS ARE LABELED R0,R1...R15. IF YOU WANT, YOU CAN THINK OF THEM AS "BASIC" VARIABLES. INFORMATION IS STORED IN THEM FOR SAFE KEEPING, AND LATER USED IN A VARIETY OF WAYS.

ONE THING THAT A REGISTER IS GOOD FOR IS HOLDING A RETURN ADDRESS FROM A SUBROUTINE CALL. WHEN THE TI DOES A "SIMPLE" SUBROUTINE CALL, (BL: BRANCH & LINK) IT PUTS THE ADDRESS OF THE NEXT INSTRUCTION INTO REGISTER R11. WHEN THE SUBROUTINE IS DONE, ALL THAT IS NECESSARY TO DO IS TO BRANCH TO THE ADDRESS IN R11. THE MACHINE LANGUAGE INSTRUCTION FOR THIS WOULD BE:

```
          B    *R11
```

THE STAR IN FRONT OF THE R11 TELLS THAT THE INFORMATION IN THE REGISTER IS AN ADDRESS, NOT DATA OR A PROGRAM. THIS KIND OF BRANCHING IS CALLED, **INDIRECT**. THE REASON IS THAT WE ARE NOT BRANCHING DIRECTLY TO R11 BUT INSTEAD WE USE R11 TO TELL US WHERE TO GO.

SAMPLE PROGRAM:

JUST TO SHOW YOU THAT MACHINE LANGUAGE REALLY WORKS, WE WILL WRITE THE SIMPLEST PROGRAM. GO TO THE MAIN MENU, TYPE:

2	: TO GET TO EASY BUG
ENTER	: TO GET TO COMMAND LEVEL
M7D00 **ENTER**	:GO TO MODIFY MODE STARTING AT >7D00
04 **ENTER**	:GIVE MEMORY LOCATION >7D00 THE VALUE >04
5B **ENTER**	:GIVE LOCATION >7D01 THE VALUE >5B
.	:CANCEL MODIFY MODE
E7D00 **ENTER**	:EXECUTE A MACHINE PROGRAM STARTING AT >7D00

IF YOU GOT ANOTHER QUESTION MARK, YOU DID EVERY THING RIGHT. THE PROGRAM THAT WE JUST WROTE IS:

```
        B     *R11
```

WHEN WE TOLD EASY BUG TO EXECUTE OUR PROGRAM (E7D00), IT CAUSED A BRANCH AND LINK (" BL @>7D00") TO OUR SUBROUTINE. ALL WE DID WAS TO BRANCH BACK. NOW WE KNOW HOW TO EXECUTE A MACHINE LANGUAGE PROGRAM AND RETURN BACK

WHEN WE ENTERED OUR PROGRAM, WE MODIFIED CENTRAL PROCESSING UNIT (CPU) RAM. CPU RAM IS WHERE ALL MACHINE LANGUAGE PROGRAMS ARE PUT.

AS LONG AS WE ARE IN EASY BUG, LETS TRY ONE MORE OF ITS FEATURES. VIDEO DISPLAY PROCESSOR (VDP) RAM IS THE RAM THAT CONTAINS THE VALUES OF WHAT IS DISPLAYED ON THE SCREEN. VDP RAM LOCATION >0130 CORRESPONDS TO A SPOT IN THE MIDDLE OF THE SCREEN ABOUT ONE THIRD OF THE WAY DOWN (SEE APPENDIX II). NOW THERE IS A >20, THE HEX VALUE FOR A SPACE, AT THAT LOCATION. IN THE EXAMPLE BELOW, WE CHANGE IT TO >41, THE CODE FOR AN "A". TYPE:

.
V0130 **ENTER**
41 **ENTER**

WHAT HAPPENS IF WE TYPE ANOTHER "41 **ENTER**"? (HINT: WE ARE PUTTING IT INTO THE NEXT SCREEN LOCATION - BUT - THE SCREEN HAS SCROLLED SINCE THE LAST TIME).

LESSON II

REGISTERS ARE NO GOOD UNLESS WE CAN PUT INFORMATION INTO THEM. IN THIS LESSON YOU WILL LEARN HOW TO DO JUST THAT. FOR EXAMPLE, IF WE WANT TO PUT THE NUMBER >0123 INTO R0 WE COULD DO THAT BY:

```
        LI   R0,>0123
```

THIS SAYS LOAD IMMEDIATE R0 WITH THE VALUE >0123. ANOTHER WAY TO FILL A REGISTER IS TO PUT A COPY OF A DIFFERENT REGISTER INTO IT. AN INSTRUCTION FOR THIS IS:

```
        MOV  R0,R1
```

THIS SAYS TO MOVE A COPY OF R0 INTO R1. THE INSTRUCTION LEAVES R0 INTACT. THIS INSTRUCTION YOU WILL BE USING OFTEN. MACHINE LANGUAGE PROGRAMS ARE GENERALLY FULL OF DATA TRANSFERS OF ONE KIND OR ANOTHER.

DID YOU NOTICE THAT IN THE FIRST EXAMPLE THE DATA WENT FROM THE RIGHT OPERAND TO THE LEFT ONE? THIS IS VERY TYPICAL OF AN "IMMEDIATE" TYPE INSTRUCTION. IN THE SECOND EXAMPLE, THE DATA MOVED FROM THE LEFT OPERAND TO THE RIGHT. THIS IS THE WAY MOST OTHER INSTRUCTIONS WORK.

THE WAY TO CALL MANY OF THE TI'S SYSTEM SUBROUTINES IS TO USE THE "BLWP" INSTRUCTION. THIS STANDS FOR BRANCH AND LOAD THE WORKSPACE POINTER. WHAT THIS INSTRUCTION DOES WILL BE COVERED LATER.

NOW WE CAN WRITE ANOTHER PROGRAM:

```
        LI   R0,>0130
        LI   R1,>4100
        BLWP @>6024
        B    *R11
```

THIS TIME WE WILL INPUT IT INTO THE COMPUTER USING THE LINE-BY-LINE ASSEMBLER PROGRAM. GO TO THE MAIN MENU, TYPE "3" TO GET TO MINI-MEM. TYPE "2" TO "RUN". TYPE "NEW" IN RESPONSE TO THE PROGRAM PROMPT. FOLLOW THE INSTRUCTIONS BELOW. MAKE SURE TO TYPE AT LEAST ONE SPACE AT THE BEGINNING OF EACH LINE. THE SPACE GOES IN THE **LABEL** FIELD. THIS IS BECAUSE SO FAR WE

HAVE HAD NO NEED FOR A LABEL.

```
    AORG >7D00 ENTER
    LI R0,>0130 ENTER
    LI R1,>4100 ENTER
    BLWP @>6024 ENTER
    B *R11 ENTER
    END ENTER
ENTER
```

IF YOU DID NOT GET THE MESSAGE "0000 UNRESOLVED REFERENCES", GO BACK AND CHECK WHAT YOU TYPED. SOMETIMES YOU CAN CORRECT YOUR MISTAKE, SOMETIMES YOU WILL HAVE TO START OVER WITH "NEW".

GO TO EASY BUG AND DO AN "E7D00". AN "A" SHOULD APPEAR ON THE SCREEN AND ANOTHER "?" SHOULD APPEAR.

IN THIS PROGRAM WE USED A SYSTEM UTILITY CALLED **VSBW.** THIS ROUTINE MOVES A SINGLE CHARACTER TO THE SCREEN. FOR MORE INFORMATION SEE PAGE 35 MINI-MEM OWNER'S MANUAL. IN THE MINIMEM ENVIRONMENT THIS ROUTINE IS LOCATED AT MEMORY LOCATION >6024.

WHEN USING THE "LINE-BY-LINE ASSEMBLER", THE "R" IN FRONT OF REGISTER NUMBERS IS OPTIONAL, THOUGH HIGHLY RECOMMENDED FOR EASE OF READING. MANY INSTRUCTIONS CAN HAVE EITHER A REGISTER OR AN ABSOLUTE MEMORY LOCATION AS AN OPERAND. TO HELP THE ASSEMBLER TELL THEM APART, WE MUST PUT AN "@" IN FRONT OF A NUMBER IF IT IS TO INDICATE AN ABSOLUTE MEMORY LOCATION.

ADVANCED EXAMPLE:

```
         AORG >7D00
         LI   R0,>0045
         LI   R1,S
         LI   R2,>000E
         BLWP @>6028
         B    *R11
S        TEXT 'THIS IS A TEST'
         SYM
         END
```

THIS EXAMPLE USES A ROUTINE CALLED **VMBW** WHICH DOES A MULTI-BYTE WRITE TO VDP RAM. IT ALSO MAKES USE OF A LABEL.

LESSON III

THE THING THAT COMPUTERS DO BEST IS DOING THE SAME THING OVER AND OVER AND OVER AGAIN. SO FAR WE HAVE BEEN HAVING IT DO ONE THING ONCE. NOW WE'LL MAKE IT DO SOME REAL WORK. LET'S HAVE THE COMPUTER FILL THE SCREEN WITH "A"S. THE PROGRAM WOULD BE:

```
         AORG >7D00
         LI   R0,>02FF
         LI   R1,>4100
L        BLWP @>6024
         DEC  R0
         JOC  L
         B    *R11
         END
```

USE "NEW" TO ENTER THIS PROGRAM. USE EASY BUG TO EXECUTE IT. THIS PROGRAM WILL FILL THE SCREEN FROM THE BOTTOM TO THE TOP. THE LOOP WILL EXECUTE EXACTLY >0300 TIMES. THE INSTRUCTION THAT CAUSES THE LOOPING IS " JOC L". "JOC" STANDS FOR JUMP ON CARRY. THE CARRY FLAG IS ONE OF THE BITS OF THE STATUS REGISTER. THE STATUS REGISTER IS NOT ONE OF YOUR WORKSPACE REGISTERS. THE CARRY FLAG IS CONDITIONED ANY TIME ANYONE DOES AN ARITHMETIC OPERATION. THE OPERATION THAT WE DID WAS DEC. "DEC" STANDS FOR DECREMENT. " DEC R0" TELLS THE COMPUTER TO SUBTRACT ONE FROM R0. IF R0 IS NOT ZERO, THE CARRY FLAG WILL BE SET TO "1", THAT IS, THERE WILL BE A CARRY. IF R0 IS ZERO, WHEN WE TRY TO SUBTRACT, WE WILL HAVE TO BORROW ONE TO DO IT. WE BORROW IT FROM THE CARRY FLAG. THEREFORE THE CARRY FLAG WILL NO LONGER BE SET; THERE WILL BE NO CARRY. WHEN THERE IS NO CARRY, THE LOOP WILL BE DONE, WE WILL DROP OUT OF IT, AND BRANCH BACK TO EASY BUG. FOR MORE INFORMATION ON THE STATUS REGISTER AND THE STATUS BITS, SEE PAGE 40 OF THE EDITOR/ASSEMBLER OWNER'S MANUAL.

ANOTHER WAY TO FILL THE SCREEN WOULD BE FROM THE TOP DOWN. THAT PROGRAM WOULD BE:

```
          AORG >7D00
          CLR  R0
          LI   R1,>4100
L         BLWP @>6024
          INC  R0
          CI   R0,>0300
          JNE  L
          B    *R11
          END
```

" CLR R0" STANDS FOR CLEAR R0. WHAT THIS DOES IS TO SET THE WHOLE WORD OF R0 TO ZERO. THIS IS AN ABREVIATION FOR " LI R0,>0000". " INC R0" SAYS TO INCREMENT R0 (BY ONE). WE WANT THIS LOOP TO START AT ZERO, THE FIRST LOCATION ON THE SCREEN. WE KNOW WE ARE DONE WHEN R0 IS EQUAL TO >0300. SO WE (" CI R0,>0300") COMPARE IMMEDIATE R0 WITH >0300. AND WE (" JNE L" JUMP (WHILE) NOT EQUAL TO L.

ADVANCED EXAMPLE:

```
    AORG >7D00
    CLR  R0                :1 WHERE TO PRINT
    LI   R1,>4100          :2 WHAT TO PRINT
    LI   R2,>02FF          :3 HOW MANY TO PRINT
    ORI  R0,>4000          :4
    SWPB R0                :5
    MOVB R0,@>8C02         :6 LOW BYTE
    SWPB R0                :7
    MOVB R0,@>8C02         :8 HI BYTE
L   MOVB R1,@>8C00         :9
    DEC  R2                :10
    JNE  L                 :11
    B    *R11              :12
TX  TEXT '  PRINT THIS'    :13 USED IN THE NEXT EXAMPLE
    END
```

"ORI" IS "OR" IMMEDIATE. "SWPB" IS SWAP BYTES. "SWPB" IS USED TO EXCHANGE THE BYTES IN A WORD WITH EACH OTHER. IN THIS CASE IT IS USED TO KILL SOME TIME AND ALSO TO PUT THE PROPER BYTE IN THE FIRST POSITION. LINES 4-8 SET UP A WRITE TO VDP RAM STARTING AT THE LOCATION SPECIFIED IN R0. FOR MORE INFORMATION SEE PAGE 266 OF THE <u>EDITOR/ASSEMBLER OWNER'S MANUAL</u>.

```
7D00        CLR  R0                :1
7D02        LI   R1,>7D24          :2
7D06        LI   R2,>000C          :3
7D0A        ORI  R0,>4000          :4
7D0E        SWPB R0                :5
7D10        MOVB R0,@>8C02         :6
7D14        SWPB R0                :7
7D16        MOV  R0,@>8C02         :8
7D1A L      MOVB *R1+,@>8C00       :9
7D1E        DEC  R2                :10
7D20        JNE  L                 :11
7D22        B    *R11              :12
7D24 TX     TEXT '  PRINT THIS'    :13
            END
```

THE UNDERLINED LINES ARE THE ONLY ONES THAT ARE DIFFERENT FROM PREVIOUS EXAMPLE. TO CHANGE THEM YOU COULD RETYPE THE WHOLE PROGRAM OR YOU COULD USE **AORG** COMMAND TO SET THE LOCATION COUNTER TO THE ADDRESS OF THE LINE YOU WANT TO CHANGE. AFTER YOU HAD CHANGED THE COUNTER, YOU CAN ENTER THE NEW FORM OF THE LINE. AN EXAMPLE OF HOW TO DO THIS WOULD BE:

```
AORG >7D02
LI   R1,>7D24
LI   R2,>000A
AORG >7D1A
MOVB *R1+,@>8C00
END
```

IN LESSON ONE WE LEARNED HOW TO USE INDIRECT ADDRESSING WITH A BRANCH COMMAND. LINE #9 IS AN EXAMPLE OF USING IT WITH A MOVE COMMAND. IF YOU REMEMBER, WHEN WE USE INDIRECT ADDRESSING WE PUT THE ADDRESS OF THE OPERAND INTO THE REGISTER. THIS EXAMPLE IS DIFFERENT IN THAT IT ALSO ILLUSTRATES **AUTO-INCREMENTING.** AUTO-INCREMENTING MEANS THAT EACH TIME WE FINISH EXECUTING THE INSTRUCTION, THE VALUE IN THE REGISTER IS INCREMENTED. IN OUR EXAMPLE, BECAUSE WE WERE MOVING BYTES, THE REGISTER IS INCREMENTED BY ONE. IF WE USE AUTO-INCREMENT WITH AN INSTRUCTION THAT INVOLVES WORDS, THE REGISTER IS INCREMENTED BY TWO.

LESSON IV

MANY TIMES THE FLOW OF CONTROL OF A PROGRAM IS NOT LINEAR. SOMETIMES ALL THAT IS NEEDED IS A LOOP, BUT SOMETIMES WHAT IS CALLED FOR IS A JUMP TO A SUBROUTINE. SUBROUTINES ARE SEGMENTS OF CODE THAT ARE NOT IN THE MAIN STREAM OF THE PROGRAM. THEY MAY BE AT THE BEGINNING OR AT THE END. THE REASONS FOR USING SUBROUTINES IN MACHINE LANGUAGE ARE MUCH THE SAME AS IN BASIC. IT MAY BE TO MAKE THE PROGRAM EASIER TO READ, OR MAYBE BECAUSE THAT PIECE OF CODE IS USED BY DIFFERENT PARTS OF THE PROGRAM. ONE KIND OF SUBROUTINE CALL IS "BL". "BL" STANDS FOR BRANCH AND LINK. WHEN WE DO A BRANCH AND LINK, THE COMPUTER SAVES THE ADDRESS OF THE STATEMENT AFTER THE "CALL". THAT ADDRESS TELLS THE SUBROUTINE WHERE TO GO WHEN IT IS DONE. THIS INSTRUCTION PUTS THE RETURN ADDRESS INTO R11. VERY OFTEN WE HAVE TO SAVE THIS VALUE SOMEWHERE ELSE SO THAT FURTHER BRANCHING AND LINKING CAN TAKE PLACE. HERE IS AN EXAMPLE THAT PRINTS AN "A" AT A GIVEN X AND Y COORDINATE:

```
          AORG >7D00
          MOV  R11,R10         :1
          LI   R4,>0010        :2
          LI   R5,>0015        :3
          LI   R1,>4100        :4
          BL   @XY             :5
          B    *R10            :6
XY        MOV  R5,R0           :7
          SLA  R0,5            :8
          A    R4,R0           :9
          BLWP @>6024          :10
          B    *R11            :11
          END
```

LINE 1: THIS LINE SAVES THE LINK GENERATED BY EASY BUG'S CALL TO OUR SUBROUTINE. WE PUT IT INTO R10.

LINE 2: R4 IS THE X COORDINATE OF WHERE WE WILL PRINT AN "A"

LINE 3: R5 IS THE Y CO-ORDINATE

LINE 4: LOAD R1 WITH AN "A"

LINE 5: BRANCH AND LINK TO OUR PRINT SUBROUTINE

LINE 6: RETURN TO EASY BUG.

LINE 7: COPY R5 INTO R0

LINE 8: SHIFT LEFT ARITHMETIC (" SLA R0"). EVERY TIME A WORD IS SHIFTED ONE PLACE LEFT, IT IS EFFECTIVELY MULTIPLIED BY 2. SHIFTING IT LEFT 5 PLACES WILL MULTIPLY IT BY 32.

LINE 9: ADD (" A R4,R0") R4 TO R0. AT THIS POINT R0=32*Y+X

LINE 10: PRINT AN "A" AT THE LOCATION WE CALCULATED

LINE 11: RETURN BACK TO LINE 6

TYPE THIS PROGRAM IN. EXECUTE IT. NOW TRY TO SAVE IT. CONNECT YOUR TAPE RECORDER. TYPE S7D00 **ENTER.** THIS TELLS EASY-BUG TO SAVE MEMORY STARTING AT LOCATION >7D00. WHEN IT ASKS FOR "TO", TYPE 7D20. THIS TELLS IT TO SAVE THROUGH >7D20. FOLLOW THE INSTRUCTIONS ON THE SCREEN. TO CHECK IF IT WORKED, GO TO MODIFY MODE AND PUT >00"S IN MEMORY STARTING AT >7D00. NOW LOAD THE PROGRAM BACK IN AND SEE IF YOU CAN STILL EXECUTE IT. SINCE THERE WILL BE WRITING ON THE SCREEN ALREADY, FINDING THE NEW "A" MAY BE A LITTLE BIT TRICKY

EXERCISE:

```
            AORG >7D00          :DRIVER ROUTINE
            LWPI >70B8          :SEE LESSON 5
            CLR  @>8374         :CLEAR KEYBOARD SELECT
            LI   R8,>1000       :SET SPEED OF PADDLE
D           MOV  R8,R7
            BL   @P             :CALL PADDLE ROUTINE
D1          DEC  R7             :DELAY LOOP
            JNE  D1
            JMP  D


            AORG >7E00          :MOVING PADDLE ROUTINE
P           MOV  R11,R9         :SAVE RETURN
            CLR  R3
            LI   R1,P6          :LOAD R1 WITH A BLANK PADDLE
            BL   @P4            :ERASE PADDLE
            BLWP @>6020         :CALL KEYSCAN
            MOVB @>8375,R3      :MOVE ASCII BYTE INTO R3
            ORI  R3,>2000       :MASK TO TURN UPPER CASE INTO LOWER
            CI   R3,>6400       :CHECK FOR  "d"
            JEQ  P1             :IF FOUND JUMP TO MOVE RIGHT
            CI   R3,>7300       :CHECK FOR "s"
            JEQ  P2             :IF FOUND JUMP TO MOVE LEFT
            JMP  P3             :JUMP TO PRINT
P1          CI   R6,>0019       :CHECK IF ALL THE WAY RIGHT
            JEQ  P3
            INC  R6
            JMP  P3
P2          CI   R6,>0002       :CHECK IF ALL THE WAY LEFT
            JEQ  P3
            DEC  R6
P3          LI   R1,P5          :LOAD R1 WITH SOLID PADDLE
            MOV  R9,R11         :"TRICK" TO GET US BACK TO DRIVER
P4          MOV  R6,R0
            AI   R0,>0280
            LI   R2,3
            BLWP @>6028
            B    *R11
P5          TEXT '---'
P6          TEXT '   '
```

ENTER AND EXECUTE (YOU WILL HAVE TO TURN OFF THE COMPUTER TO EXIT). SAVE THE "P" ROUTINE (>7E00 - >7E53). YOU WILL NEED IT LATER. IF YOU WANT TO CHECK TO SEE IF YOU TYPED IT IN RIGHT, THERE IS A LISTING IN APPENDIX 4 THAT GIVES THE ADDRESSES AND THE ASSOCIATED VALUES FOR THE "P" ROUTINE.

LESSON V

TI CALLS ITS REGISTERS **WORKSPACE REGISTERS** BECAUSE THEY CAN BE USED TO DEFINE AN ENVIRONMENT THAT GIVES SUBROUTINES A UNIQUE CONTEXT IN WHICH TO OPERATE. YOU, THE USER, HAVE THE ABILITY TO SPECIFY WHERE THE WORKSPACE REGISTERS WILL BE IN MEMORY. INFACT, YOU CAN HAVE AS MANY SETS OF REGISTERS AS YOU WANT. THE SET THAT IS CURRENTLY ACTIVE IS THE ONE POINTED TO BY THE **WORKSPACE POINTER.** WHEN YOU CHANGE WHICH SET OF REGISTERS YOU ARE USING, THIS IS REFERRED TO AS A **CONTEXT SWITCH.** ONE INSTRUCTION THAT CAUSES A CONTEXT SWITCH IS "LWPI". IN THE LAST EXAMPLE WE USED " LWPI >70B8" TO LOAD IMMEDIATE THE WORKSPACE POINTER WITH THE VALUE >70B8. THIS INSTRUCTION DESTROYS WHAT WAS IN THE POINTER SO CARE MUST BE TAKEN TO SAVE IT FIRST. THE REASON WE USED "LWPI" IN THE PREVIOUS EXAMPLE WAS BECAUSE EASY-BUG USES THE GPL WORKSPACE REGISTERS. THESE REGISTERS ARE LOCATED AT >83E0, AND ARE USED BY GPL ROUTINES. KSCAN IS A GPL ROUTINE AND WOULD CAUSE SIDE EFFECTS TO OUR PROGRAM. WE AVOID THE PROBLEM BY SETTING UP OUR OWN REGISTERS. THE ONES THAT WE USED ARE CALLED USRWSP AND ARE LOCATED AT >70B8.

ANOTHER INSTRUCTION THAT CAUSES A CONTEXT SWITCH IS "BLWP". "BLWP" STANDS FOR BRANCH AND LOAD THE WORKSPACE POINTER. TO USE A "BLWP" INSTRUCTION, YOU MUST SET UP A PAIR OF WORDS. THE FIRST WORD IS A POINTER TO A SET OF REGISTERS, THE SECOND IS AN ENTRY POINT INTO YOUR SUBROUTINE. WHEN ONE EXECUTES THIS INSTRUCTION, MANY THINGS HAPPEN. FIRST THE COMPUTER DOES A CONTEXT SWITCH, THEN IT PUTS THE OLD WP, THE OLD PC AND THE VALUE OF THE OLD STATUS REGISTER INTO THE NEW REGISTERS R13-R15. FINALLY THE COMPUTER BRANCHES TO THE SUBROUTINE.

```
          AORG >7D00                 :DRIVER
          LI   R8,>1000              :SPEED OF THE "A"
Z         MOV  R8,R7
          BLWP @M                    :MOVING "A" SUBROUTINE
Z1        DEC  R7                    :DELAY
          JNE  Z1
          JMP  Z
```

```
M        AORG >7E60
         DATA MR                    MOVING "A" ROUTINE
         DATA MM

MR       DATA >0000  R0   :VSBW ADDRESS
         DATA >0000  R1   :VSBW DATA
         DATA >0010  R2   :X
         DATA >0005  R3   :Y
         DATA >0001  R4   :X INCREMENT
         DATA >0001  R5   :Y INCREMENT
         DATA >0002  R6   :X MIN (LEFT WALL)
         DATA >0003  R7   :Y MIN (TOP WALL)
         DATA >001B  R8   :X MAX (RIGHT WALL)
         DATA >0017  R9   :Y MAX (BOTTOM WALL)
         DATA >4100  R10  :"A"
         DATA >0000  R11  :"BL" RETURN ADDRESS
         DATA >2000  R12  :" "
         DATA >0000  R13  :OLD WP
         DATA >0000  R14  :OLD PC
         DATA >0000  R15  :OLD STATUS

MM       MOV  R12,R1
         BL   @M5
         C    R2,R6             :HAS IT HIT THE LEFT WALL?
         JNE  M1
         NEG  R4                :CHANGE X DIRECTION
M1       C    R2,R8             :HIT RIGHT WALL?
         JNE  M2
         NEG  R4                :CHANGE X DIRECTION
M2       A    R4,R2             :UPDATE X POSITION
         C    R3,R7             :HIT TOP?
         JNE  M3
         NEG  R5                :CHANGE Y DIRECTION
M3       C    R3,R9             :HIT BOTTOM?
         JNE  M4
         NEG  R5                :CHANGE Y DIRECTION
M4       A    R5,R3             :UPDATE Y POSITION
         MOV  R10,R1
         BL   @M5               :CALL PRINT
         RTWP
M5       MOV  R3,R0             :PRINT AT "X","Y" (R2,R3)
         SLA  R0,5               ROUTINE
         A    R2,R0
         CI   R0,>2FF           :ERROR CHECK
         JH   M6
         BLWP @>6024
M6       B    *R11
         END
```

THE FIRST THREE LINES ARE A SHORT DRIVER PROGRAM, THEY CALL OUR SUBROUTINE AND THEN RETURN. THE NEXT TWO LINES ARE A POINTER TO OUR SET OF REGISTERS, AND A POINTER TO THE BEGINNING OF OUR SUBROUTINE. A "BLWP" TO THE FIRST OF THESE POINTERS CAUSES A CONTEXT SWITCH (CHANGING OF THE WP) AND ALSO CAUSES OUR SUBROUTINE TO BE EXECUTED. IN ADDITION, THE OLD WP, THE OLD PROGRAM COUNTER, AND THE OLD STATUS REGISTER ARE PUT INTO THE NEW REGISTERS R13,R14,R15 RESPECTIVELY.

DID YOU NOTICE THAT A LOT OF THE REGISTERS ARE ALREADY INITIALIZED. THE NICE THING ABOUT A CONTEXT SWITCH IS THAT AN ENVIRONMENT CAN BE READY FOR YOU TO GO IN AND USE.

TYPE THIS IN, RUN IT, SAVE THE "M" ROUTINE (>7E60 - >7EBF).

LESSON VI

THE BEST WAY TO LEARN THINGS IS TO EXPERIMENT. UNTIL YOU TRY SOMETHING ON YOUR OWN AND MAKE A FEW MISTAKES, YOU NEVER REALLY LEARN. UNFORTUNATELY, MACHINE LANGUAGE CAN BE VERY UNFORGIVING WHEN IT COMES TO MAKING MISTAKES. ONE AID TO WRITING AND DEBUGGING PROGRAMS IS TO USE **BREAK POINTS**. WHAT A BREAK POINT DOES IS TO CALL A ROUTINE THAT DISPLAYS SOME INFORMATION ABOUT THE STATE OF THE COMPUTER. THE ROUTINE IN THE NEXT EXAMPLE WILL DISPLAY A SPECIFIED NUMBER OF THE CALLING PROGRAM'S REGISTERS. IT CAN DISPLAY THEM IN HEXADECIMAL OR DECIMAL AND IT WILL DISPLAY THE PROGRAM COUNTER IF THAT IS SO DESIRED. WHAT THE ROUTINE DISPLAYS IS DETERMINED BY THE **PARAMETERS** YOU SEND TO IT. AFTER IT DISPLAYS ITS INFORMATION, THE ROUTINE WILL WAIT FOR YOU TO PRESS A KEY. ANY KEY BUT THE SPACE WILL STEP THROUGH THE PROGRAM ONE BREAK POINT AT A TIME. THE SPACE KEY WILL STEP CONTINUOUSLY THROUGH THE PROGRAM AS LONG AS YOU HOLD IT DOWN.

TO USE BREAK POINTS ONE MUST PLAN AHEAD. IF WE CALL THE ROUTINE WITH THE INSTRUCTION " BLWP *R9" WHERE R9 HAS THE ADDRESS OF OUR ROUTINE, WE HAVE TO ALLOW ONE WORD OF MEMORY FOR EACH PLACE WE MAY WANT TO INSERT A BREAK POINT. THE EASIEST WAY TO DO THAT IS TO USE THE "NOP" INSTRUCTION. "NOP" IS AN ASSEMBLER ABREVIATION FOR " JMP $+2", WHICH SAYS TO JUMP TO THE NEXT INSTRUCTION. THE MACHINE CODE FOR " BLWP *R9" IS >0419. THE MACHINE CODE FOR "NOP" IS >1000. IF WE EXCHANGE THESE TWO VALUES IN A LOCATION WHERE WE HAVE ALLOWED SPACE FOR A BREAK POINT WE CAN TURN THE FUNCTION ON OR OFF.

NOW TO SHOW WHAT I AM TALKING ABOUT:

```
          AORG >7D00
          LWPI >70B8
          LI   R9,>7F10
S         LI   R0,>0100
S1        NOP
          DEC  R0
          JNE  S1
          JMP  S
          END
```

IF YOU EXECUTE THIS, NOTHING WILL HAPPEN. BUT IF YOU CHANGE THE "NOP" AT >7D0C TO A " BLWP *R9" WONDEROUS THINGS WILL HAPPEN (ESPECIALLY IF YOU DON'T TYPE IN THE NEXT PROGRAM FIRST).

```
        AORG >7F10
TX      DATA TW           :BREAK POINT ROUTINE
        DATA TT
TT      BL   @T
        DATA >0096        PARAMETER #1: WHERE TO PRINT
        DATA >0000        #2: WHICH ONE TO START WITH
        DATA >0005        #3: HOW MANY
        DATA >0000        #4:IF <>0 THEN CONVERT TO DECIMAL
        DATA >0001        #5:IF <>0 THEN PRINT "PC"
        RTWP
TW      BSS  >20

T       MOV  R11,R10      :SAVE LINK
        MOV  *R10+,R4     :MOVE PARAMETERS
        MOV  *R10+,R1
        MOV  *R10+,R7
        MOV  *R10+,R8
        MOV  R13,R6       :MOVE OLD WP TO R6
T1      MOV  *R6+,R2      :GET VALUE FROM AN OLD REGISTER
        DEC  R1           :SHOULD WE PRINT THIS?
        JOC  T1
T2      MOV  R8,R8        :CONVERT TO DECIMAL?
        JEQ  T3
        BL   @C           :CALL CONVERT ROUTINE
T3      BL   @W           :CALL DISPLAY WORD ROUTINE
        AI   R4,>1C
        MOV  *R6+,R2      :GET ANOTHER REGISTER
        DEC  R7           :ARE WE DONE?
        JNE  T2
        MOV  *R10+,R0     :PRINT PC?
        JEQ  T4
        MOV  R14,R2
        BL   @W           :PRINT PC
T4      BL   @N           :CALL PAUSE
        B    *R10

W       LI   R3,4         :WRITE A WORD
W1      SRC  R2,>C        :SHIFT WORD 12 PLACES
        MOV  R2,R1
        ANDI R1,>000F     :MASK OFF LAST NIBBLE
        SRC  R1,8         :SWAP BYTES
        AI   R1,>3000     :CONVERT TO ASCII
        CI   R1,>3A00
        JL   W2
        AI   R1,>0700
W2      CI   R4,>0300     :ERROR CHECK
```

```
         JL    W3
         CLR   R4
W3       MOV   R4,R0
         INC   R4
         BLWP  @>6024
         DEC   R3
         JNE   W1
         B     *R11

N        CLR   R0                :PAUSE ROUTINE
         MOV   R0,@>8374         :CLEAR KEYBOARD SELECT
N1       BLWP  @>6020            :KEYSCAN
         MOVB  @>8375,R0         :MOVE ASCII BYTE
         CI    R0,>2000          :CHECK IF BLANK
         JEQ   N2
         MOV   @>837C,R0         :MOVE STATUS
         ANDI  R0,>2000          :CHECK IF NEW KEY
         JEQ   N1
N2       B     *R11

C        LI    R3,C2             :CONVERT HEX TO DEC
         CLR   R1
         CLR   R0
C1       DIV   *R3+,R1
         SLA   R0,4
         SOC   R1,R0
         CLR   R1
         CI    R3,C2+8
         JNE   C1
         MOV   R0,R2
         B     *R11
C2       DATA  1000,100,10,1
         END
```

ADVANCED:

```
         AORG  >7D00             :THIS ROUTINE MULTIPLIES R0 AND R1
G        CLR   R0                  AND PUTS THE RESULT IN R2 AND R3
G1       CLR   R1
G2       MOV   R1,R2
         MPY   R0,R2
         BLWP  @>7F10            :CALL TRACE ROUTINE
         INC   R1
         CI    R1,>0020
         JNE   G2
         INC   R0
         CI    R0,>0020
         JNE   G1
         JMP   G
         END
```

LESSON VII

THIS IS THE FINAL LESSON OF THIS FIRST TUTOR. I HOPE THIS EXPERIENCE HAS BEEN REWARDING AND NOT TOO FRUSTRATING. HOPEFULLY I CAN TIE ALL OF YOUR EFFORTS TOGETHER AND GIVE YOU A LITTLE GAME TO PLAY. AT THIS POINT, MINI-MEM SHOULD CONTAIN THE "P", "M", AND "W" ROUTINES. IF YOU HAVE RE-INITIALIZED MINI-MEM OR THINK ANY OF THE ROUTINES MAY HAVE BEEN DESTROYED, RETYPE OR RELOAD THEM BEFORE TYPING IN THIS LAST ROUTINE.

```
          AORG >7D00
          CLR  @>8374
          LWPI >70B8
          CLR  R3
          CLR  R7
          CLR  R8
          BLWP @I                :DRAWS A BORDER
          LI   R6,>0006          :INITIALIZE PADDLE POSITION
          BL   @S                :PRINT "SCORE"
          DATA >02D2,SC,>0005
          BL   @S                :PRINT "HI SCORE"
          DATA >02EF,HS,>0008
          LI   R4,>02F8
          CLR  R2
          BLWP @>7F80            :PRINT "0000" USING "W" ROUTINE
D         DEC  R14               :SLOW DOWN PADDLE
          JGT  D7
          BL   @>7E00
          INV  R13               :MOVE "A" HALF AS OFTEN
          JLT  D6
          BLWP @>7E60
          LI   R1,>0014
          C    @>7E6A,R1         :CHECK "A" VERTICAL POSITION
          JL   D6                  (>7E6A IS R3 IN "M" ROUTINE,
          MOV  R6,R0               HERE IT IS A MEMORY LOCATION)
          LI   R1,>0003
D4        C    R0,@>7E68         :IS "A" HITTING THE PADDLE?
          JEQ  D5
          INC  R0
          DEC  R1
          JNE  D4
          JMP  D9                :IF NOT; GAME OVER

D5        NEG  @>7E6E
D6        MOVB R8,R14            :THE SPEED OF THE "A" IS RELATED
          INV  R14                 TO THE SCORE COUNTER
          SRL  R14,6
```

```
D7      DEC  R15             :SLOW DOWN SCORE COUNTER
        JGT  D8
        LI   R15,>0080
        LI   R4,02D8
        INC  R8
        MOV  R8,R2
        NOP                  :REPLACE WITH " BL @>7FD2 FOR
        NOP                   DECIMAL SCORING
        BL   @>7F7C          :PRINT SCORE USING "W" ROUTINE
D8      JMP  D

D9      LI   R0,>0005
        MOV  R0,@>7E6A       :PUT "A" AT TOP FOR NEXT GAME
        C    R8,R7           :UPDATE "HI SCORE"
        JL   DA
        MOV  R8,R2
        MOV  R8,R7
        LI   R4,>02F8
        NOP                  :REPLACE WITH " BL @>7FD2" FOR
        NOP                   DECIMAL SCORING
        BL   @>7F7C
DA      BL   @S              :PRINT "GAME OVER ..."
        DATA >0284,OV,>0016
DB      BLWP @>6020          :KEYSCAN
        MOV  @>837C,R0
        ANDI R0,>2000
        JEQ  DB
        LI   R0,>0282
        LI   R1,>2000
        LI   R2,>001A
DC      BLWP @>6024
        INC  R0
        DEC  R2
        JNE  DC
        CLR  R8
        JMP  D

S       MOV  *R11+,R0
        MOV  *R11+,R1
        MOV  *R11+,R2
        BLWP @>6028
        B    *R11

HS      TEXT 'HI '
SC      TEXT 'SCORE'
OV      TEXT 'GAME OVER-PRESS A KEY'

        AORG >7ED0
I       DATA >7E64           :WORK SPACE FOR "M" ROUTINE
        DATA II
```

```
II      LI   R1,>2A00
        MOV  R6,R2
        DEC  R2
        MOV  R9,R3
I1      BL   @>7EAE         :PRINT ROUTINE IN "M"
        DEC  R3
        C    R7,R3
        JLE  I1
I2      BL   @>7EAE
        INC  R2
        C    R2,R8
        JLE  I2
I3      BL   @>7EAE
        INC  R3
        C    R3,R9
        JLE  I3

        LI   R2,>0003       :INITIALIZE "A" X POSITION
        LI   R3,>0005       :INITIALIZE "A" Y POSITION
        RTWP
```

APPENDIX I

SECOND DIGIT

FIRST DIGIT	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	012	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

TO CONVERT A 2 DIGIT HEXADECIMAL TO DECIMAL, FIND THE FIRST DIGIT IN THE LEFT COLUMN. FIND THE SECOND DIGIT IN THE TOP ROW. FIND WHERE THE ROW AND COLUMN INTERSECT, YOU WILL FIND YOUR NUMBER.

REVERSE THE PROCESS TO GO FROM DECIMAL TO HEXADECIMAL.

APPENDIX 2

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	000	001	002	003	004	005	006	007	008	009	00A	00B	00C	00D	00E	00F	010	011	012	013	014	015	016	017	018	019	01A	01B	01C	01D	01E	01F
1	020	021	022	023	024	025	026	027	028	029	02A	02B	02C	02D	02E	02F	030	031	032	033	034	035	036	037	038	039	03A	03B	03C	03D	03E	03F
2	040	041	042	043	044	045	046	047	048	049	04A	04B	04C	04D	04E	04F	050	051	052	053	054	055	056	057	058	059	05A	05B	05C	05D	05E	05F
3	060	061	062	063	064	065	066	067	068	069	06A	06B	06C	06D	06E	06F	070	071	072	073	074	075	076	077	078	079	07A	07B	07C	07D	07E	07F
4	080	081	082	083	084	085	086	087	088	089	08A	08B	08C	08D	08E	08F	090	091	092	093	094	095	096	097	098	099	09A	09B	09C	09D	09E	09F
5	0A0	0A1	0A2	0A3	0A4	0A5	0A6	0A7	0A8	0A9	0AA	0AB	0AC	0AD	0AE	0AF	0B0	0B1	0B2	0B3	0B4	0B5	0B6	0B7	0B8	0B9	0BA	0BB	0BC	0BD	0BE	0BF
6	0C0	0C1	0C2	0C3	0C4	0C5	0C6	0C7	0C8	0C9	0CA	0CB	0CC	0CD	0CE	0CF	0D0	0D1	0D2	0D3	0D4	0D5	0D6	0D7	0D8	0D9	0DA	0DB	0DC	0DD	0DE	0DF
7	0E0	0E1	0E2	0E3	0E4	0E5	0E6	0E7	0E8	0E9	0EA	0EB	0EC	0ED	0EE	0EF	0F0	0F1	0F2	0F3	0F4	0F5	0F6	0F7	0F8	0F9	0FA	0FB	0FC	0FD	0FE	0FF
8	100	101	102	103	104	105	106	107	108	109	10A	10B	10C	10D	10E	10F	110	111	112	113	114	115	116	117	118	119	11A	11B	11C	11D	11E	11F
9	120	121	122	123	124	125	126	127	128	129	12A	12B	12C	12D	12E	12F	130	131	132	133	134	135	136	137	138	139	13A	13B	13C	13D	13E	13F
10	140	141	142	143	144	145	146	147	148	149	14A	14B	14C	14D	14E	14F	150	151	152	153	154	155	156	157	158	159	15A	15B	15C	15D	15E	15F
11	160	161	162	163	164	165	166	167	168	169	16A	16B	16C	16D	16E	16F	170	171	172	173	174	175	176	177	178	179	17A	17B	17C	17D	17E	17F
12	180	181	182	183	184	185	186	187	188	189	18A	18B	18C	18D	18E	18F	190	191	192	193	194	195	196	197	198	199	19A	19B	19C	19D	19E	19F
13	1A0	1A1	1A2	1A3	1A4	1A5	1A6	1A7	1A8	1A9	1AA	1AB	1AC	1AD	1AE	1AF	1B0	1B1	1B2	1B3	1B4	1B5	1B6	1B7	1B8	1B9	1BA	1BB	1BC	1BD	1BE	1BF
14	1C0	1C1	1C2	1C3	1C4	1C5	1C6	1C7	1C8	1C9	1CA	1CB	1CC	1CD	1CE	1CF	1D0	1D1	1D2	1D3	1D4	1D5	1D6	1D7	1D8	1D9	1DA	1DB	1DC	1DD	1DE	1DF
15	1E0	1E1	1E2	1E3	1E4	1E5	1E6	1E7	1E8	1E9	1EA	1EB	1EC	1ED	1EE	1EF	1F0	1F1	1F2	1F3	1F4	1F5	1F6	1F7	1F8	1F9	1FA	1FB	1FC	1FD	1FE	1FF
16	200	201	202	203	204	205	206	207	208	209	20A	20B	20C	20D	20E	20F	210	211	212	213	214	215	216	217	218	219	21A	21B	21C	21D	21E	21F
17	220	221	222	223	224	225	226	227	228	229	22A	22B	22C	22D	22E	22F	230	231	232	233	234	235	236	237	238	239	23A	23B	23C	23D	23E	23F
18	240	241	242	243	244	245	246	247	248	249	24A	24B	24C	24D	24E	24F	250	251	252	253	254	255	256	257	258	259	25A	25B	25C	25D	25E	25F
19	260	261	262	263	264	265	266	267	268	269	26A	26B	26C	26D	26E	26F	270	271	272	273	274	275	276	277	278	279	27A	27B	27C	27D	27E	27F
20	280	281	282	283	284	285	286	287	288	289	28A	28B	28C	28D	28E	28F	290	291	292	293	294	295	296	297	298	299	29A	29B	29C	29D	29E	29F
21	2A0	2A1	2A2	2A3	2A4	2A5	2A6	2A7	2A8	2A9	2AA	2AB	2AC	2AD	2AE	2AF	2B0	2B1	2B2	2B3	2B4	2B5	2B6	2B7	2B8	2B9	2BA	2BB	2BC	2BD	2BE	2BF
22	2C0	2C1	2C2	2C3	2C4	2C5	2C6	2C7	2C8	2C9	2CA	2CB	2CC	2CD	2CE	2CF	2D0	2D1	2D2	2D3	2D4	2D5	2D6	2D7	2D8	2D9	2DA	2DB	2DC	2DD	2DE	2DF
23	2E0	2E1	2E2	2E3	2E4	2E5	2E6	2E7	2E8	2E9	2EA	2EB	2EC	2ED	2EE	2EF	2F0	2F1	2F2	2F3	2F4	2F5	2F6	2F7	2F8	2F9	2FA	2FB	2FC	2FD	2FE	2FF

THIS TABLE SHOWS HOW VDP MEMORY MAPS ONTO THE TV SCREEN

APPENDIX 3

ASCII CODES

>20	SPACE	>40	@	>60	`
>21	!	>41	A	>61	a
>22	"	>42	B	>62	b
>23	#	>43	C	>63	c
>24	$	>44	D	>64	d
>25	%	>45	E	>65	e
>26	&	>46	F	>66	f
>27	'	>47	G	>67	g
>28	(	>48	H	>68	h
>29	)	>49	I	>69	i
>2A	*	>4A	J	>6A	j
>2B	+	>4B	K	>6B	k
>2C	,	>4C	L	>6C	l
>2D	-	>4D	M	>6D	m
>2E	.	>4E	N	>6E	n
>2F	/	>4F	O	>6F	o
>30	0	>50	P	>70	p
>31	1	>51	Q	>71	q
>32	2	>52	R	>72	r
>33	3	>53	S	>73	s
>34	4	>54	T	>74	t
>35	5	>55	U	>75	u
>36	6	>56	V	>76	v
>37	7	>57	W	>77	w
>38	8	>58	X	>78	x
>39	9	>59	Y	>79	y
>3A	:	>5A	Z	>7A	z
>3B	;	>5B	[	>7B	{
>3C	<	>5C	\	>7C	
>3D	=	>5D	]	>7D	}
>3E	>	>5E	^	>7E	~
>3F	?	>5F	_		

```
7D00                   AORG >7D00
7D00 04E0  G           CLR  @>8374
7D02 8374
7D04 02E0              LWPI >70B8
7D06 70B8
7D08 04C3              CLR  R3
7D0A 04C7              CLR  R7
7D0C 04C8              CLR  R8
7D0E 0420              BLWP @I
7D10 7ED0
7D12 0206              LI   R6,>0006
7D14 0006
7D16 06A0              BL   @S
7D18 7DD4
7D1A 02D2              DATA >02D2,SC,>5
7D1C 7DE3
7D1E 0005
7D20 06A0              BL   @S
7D22 7DD4
7D24 02EF              DATA >02EF,HS,>8
7D26 7DE0
7D28 0008
7D2A 0204              LI   R4,>2F8
7D2C 02F8
7D2E 04C2              CLR  R2
7D30 06A0              BL   @W
7D32 7F7C
7D34 060E  D           DEC  R14
7D36 151A              JGT  D7
7D38 06A0              BL   @P
7D3A 7E00
7D3C 054D              INV  R13
7D3E 1113              JLT  D6
7D40 0420              BLWP @M
7D42 7E60
7D44 0201              LI   R1,>0014
7D46 0014
7D48 8060              C    @BY,R1
7D4A 7E6A
7D4C 1A0C              JL   D6
7D4E C006              MOV  R6,R0
7D50 0201              LI   R1,3
7D52 0003
7D54 8800  D4          C    R0,@BX
7D56 7E68
7D58 1304              JEQ  D5
7D5A 0580              INC  R0
7D5C 0601              DEC  R1
7D5E 16FA              JNE  D4
7D60 1012              JMP  D9
```

```
7D62 0520  D5    NEG  @IY
7D64 7E6E
7D66 D388  D6    MOVB R8,R14
7D68 054E        INV  R14
7D6A 096E        SRL  R14,6
7D6C 060F  D7    DEC  R15
7D6E 150A        JGT  D8
7D70 020F        LI   R15,>0080
7D72 0080
7D74 0204        LI   R4,>02D8
7D76 02D8
7D78 0588        INC  R8
7D7A C088        MOV  R8,R2
7D7C 1000        NOP
7D7E 1000        NOP
7D80 06A0        BL   @W
7D82 7F7C
7D84 10D7  D8    JMP  D

7D86 0200  D9    LI   R0,5
7D88 0005
7D8A C800        MOV  R0,@BY
7D8C 7E6A
7D8E 81C8        C    R8,R7
7D90 1A08        JL   DA
7D92 C088        MOV  R8,R2
7D94 C1C8        MOV  R8,R7
7D96 0204        LI   R4,>2F8
7D98 02F8
7D9A 1000        NOP
7D9C 1000        NOP
7D9E 06A0        BL   @W
7DA0 7F7C
7DA2 06A0  DA    BL   @S
7DA4 7DD4
7DA6 0284        DATA >0284,OV,>16
7DA8 7DE8
7DAA 0016
7DAC 0420  DB    BLWP @>6020
7DAE 6020
7DB0 C020        MOV  @>837C,R0
7DB2 837C
7DB4 0240        ANDI R0,>2000
7DB6 2000
7DB8 13F9        JEQ  DB
7DBA 0200        LI   R0,>282
7DBC 0282
7DBE 0201        LI   R1,>2000
7DC0 2000
7DC2 0202        LI   R2,>1A
7DC4 001A
7DC6 0420  DC    BLWP @>6024
7DC8 6024
7DCA 0580        INC  R0
7DCC 0602        DEC  R2
```

```
7DCE 16FB          JNE  DC
7DD0 04C8          CLR  R8
7DD2 10B0          JMP  D

7DD4 C03B   S      MOV  *R11+,R0
7DD6 C07B          MOV  *R11+,R1
7DD8 C0BB          MOV  *R11+,R2
7DDA 0420          BLWP @>6028
7DDC 6028
7DDE 045B          B    *R11

7DE0   48   HS     TEXT 'HI '
7DE3   53   SC     TEXT 'SCORE'
7DE8   47   OV     TEXT 'GAME OVER-PRESS A KEY '

7E00               AORG >7E00
7E00 C24B   P      MOV  R11,R9
7E02 04C3          CLR  R3
7E04 0201          LI   R1,P6
7E06 7E51
7E08 06A0          BL   @P4
7E0A 7E3E
7E0C 0420          BLWP @>6020
7E0E 6020
7E10 D0E0          MOVB @>8375,R3
7E12 8375
7E14 0263          ORI  R3,>2000
7E16 2000
7E18 0283          CI   R3,>6400
7E1A 6400
7E1C 1304          JEQ  P1
7E1E 0283          CI   R3,>7300
7E20 7300
7E22 1306          JEQ  P2
7E24 1009          JMP  P3
7E26 0286   P1     CI   R6,>0019
7E28 0019
7E2A 1306          JEQ  P3
7E2C 0586          INC  R6
7E2E 1004          JMP  P3
7E30 0286   P2     CI   R6,>0002
7E32 0002
7E34 1301          JEQ  P3
7E36 0606          DEC  R6
7E38 0201   P3     LI   R1,P5
7E3A 7E4E
7E3C C2C9          MOV  R9,R11
7E3E C006   P4     MOV  R6,R0
7E40 0220          AI   R0,>0280
7E42 0280
7E44 0202          LI   R2,3
7E46 0003
7E48 0420          BLWP @>6028
7E4A 6028
7E4C 045B          B    *R11
```

```
7E4E    2D   P5     TEXT '---'
7E51    20   P6     TEXT '   '

7E60                AORG >7E60
7E60 7E64    M      DATA MR
7E62 7E84           DATA MM

                    EVEN
7E64 0000    MR     DATA >0000
7E66 0000           DATA >0000
7E68 0010    BX     DATA >0010
7E6A 0005    BY     DATA >0005
7E6C 0001    IX     DATA >0001
7E6E 0001    IY     DATA >0001
7E70 0002           DATA >0002
7E72 0003           DATA >0003
7E74 001B           DATA >001B
7E76 0017           DATA >0017
7E78 4100           DATA >4100
7E7A 0000           DATA >0000
7E7C 2000           DATA >2000
7E7E 0000           DATA >0000
7E80 0000           DATA >0000
7E82 0000           DATA >0000

7E84 C04C    MM     MOV  R12,R1
7E86 06A0           BL   @M5
7E88 7EAE
7E8A 8182           C    R2,R6
7E8C 1601           JNE  M1
7E8E 0504           NEG  R4
7E90 8202    M1     C    R2,R8
7E92 1601           JNE  M2
7E94 0504           NEG  R4
7E96 A084    M2     A    R4,R2
7E98 81C3           C    R3,R7
7E9A 1601           JNE  M3
7E9C 0505           NEG  R5
7E9E 8243    M3     C    R3,R9
7EA0 1601           JNE  M4
7EA2 0505           NEG  R5
7EA4 A0C5    M4     A    R5,R3
7EA6 C04A           MOV  R10,R1
7EA8 06A0           BL   @M5
7EAA 7EAE
7EAC 0380           RTWP

7EAE C003    M5     MOV  R3,R0
7EB0 0A50           SLA  R0,5
7EB2 A002           A    R2,R0
7EB4 0280           CI   R0,>02FF
7EB6 02FF
7EB8 1B02           JH   M6
7EBA 0420           BLWP @>6024
7EBC 6024
```

```
7EBE 045B  M6     B     *R11


7ED0              AORG  >7ED0
7ED0 7E64  I      DATA  >7E64
7ED2 7ED4         DATA  II

7ED4 0201  II     LI    R1,>2A00
7ED6 2A00
7ED8 C086         MOV   R6,R2
7EDA 0602         DEC   R2
7EDC C0C9         MOV   R9,R3
7EDE 06A0  I1     BL    @M5
7EE0 7EAE
7EE2 0603         DEC   R3
7EE4 80C7         C     R7,R3
7EE6 12FB         JLE   I1
7EE8 06A0  I2     BL    @M5
7EEA 7EAE
7EEC 0582         INC   R2
7EEE 8202         C     R2,R8
7EF0 12FB         JLE   I2
7EF2 06A0  I3     BL    @M5
7EF4 7EAE
7EF6 0583         INC   R3
7EF8 8243         C     R3,R9
7EFA 12FB         JLE   I3

7EFC 0202         LI    R2,>3
7EFE 0003
7F00 0203         LI    R3,>5
7F02 0005
7F04 0380         RTWP

7F10              AORG  >7F10
7F10 7F24  TX     DATA  TW
7F12 7F14         DATA  TT

7F14 06A0  TT     BL    @T
7F16 7F44
7F18 0096         DATA  >0096,0,5,0,1
7F1A 0000
7F1C 0005
7F1E 0000
7F20 0001
7F22 0380         RTWP

7F24       TW     BSS   >20

7F44 C28B  T      MOV   R11,R10
7F46 C13A         MOV   *R10+,R4
7F48 C07A         MOV   *R10+,R1
7F4A C1FA         MOV   *R10+,R7
7F4C C23A         MOV   *R10+,R8
7F4E C18D         MOV   R13,R6
```

```
7F50 C0B6   T1    MOV  *R6+,R2
7F52 0601         DEC  R1
7F54 18FD         JOC  T1
7F56 C208   T2    MOV  R8,R8
7F58 1302         JEQ  T3
7F5A 06A0         BL   @C
7F5C 7FCE
7F5E 06A0   T3    BL   @W
7F60 7F7C
7F62 0224         AI   R4,>1C
7F64 001C
7F66 C0B6         MOV  *R6+,R2
7F68 0607         DEC  R7
7F6A 16F5         JNE  T2
7F6C C03A         MOV  *R10+,R0
7F6E 1303         JEQ  T4
7F70 C08E         MOV  R14,R2
7F72 06A0         BL   @W
7F74 7F7C
7F76 06A0   T4    BL   @N
7F78 7FAE
7F7A 045A         B    *R10


7F7C 0203   W     LI   R3,4
7F7E 0004
7F80 0BC2   W1    SRC  R2,>C
7F82 C042         MOV  R2,R1
7F84 0241         ANDI R1,>000F
7F86 000F
7F88 0B81         SRC  R1,8
7F8A 0221         AI   R1,>3000
7F8C 3000
7F8E 0281         CI   R1,>3A00
7F90 3A00
7F92 1A02         JL   W2
7F94 0221         AI   R1,>0700
7F96 0700
7F98 0284   W2    CI   R4,>0300
7F9A 0300
7F9C 1A01         JL   W3
7F9E 04C4         CLR  R4
7FA0 C004   W3    MOV  R4,R0
7FA2 0584         INC  R4
7FA4 0420         BLWP @>6024
7FA6 6024
7FA8 0603         DEC  R3
7FAA 16EA         JNE  W1
7FAC 045B         B    *R11

7FAE 04C0   N     CLR  R0
7FB0 C800         MOV  R0,@>8374
7FB2 8374
7FB4 0420   N1    BLWP @>6020
7FB6 6020
```

```
7FB8 D020          MOVB @>8375,R0
7FBA 8375
7FBC 0280          CI   R0,>2000
7FBE 2000
7FC0 1305          JEQ  N2
7FC2 C020          MOV  @>837C,R0
7FC4 837C
7FC6 0240          ANDI R0,>2000
7FC8 2000
7FCA 13F4          JEQ  N1
7FCC 045B  N2      B    *R11

7FCE 0203  C       LI   R3,C2
7FD0 7FE8
7FD2 04C1          CLR  R1
7FD4 04C0          CLR  R0
7FD6 3C73  C1      DIV  *R3+,R1
7FD8 0A40          SLA  R0,4
7FDA E001          SOC  R1,R0
7FDC 04C1          CLR  R1
7FDE 0283          CI   R3,C2+8
7FE0 7FF0
7FE2 16F9          JNE  C1
7FE4 C080          MOV  R0,R2
7FE6 045B          B    *R11
7FE8 03E8  C2      DATA 1000,100,10,1
7FEA 0064
7FEC 000A
7FEE 0001


                   END
```

Truth Table
for AND

	0	1
0	0	0
1	0	1

Examples:

1100 1101 = CD	1010 0001 = A1	1011 1000 = B8
0000 1111 = 0F	1001 1000 = 98	1111 0001 = F1
0000 1101 = 0D	1000 0000 = 80	1011 0000 = B0

Truth Table
for OR

	0	1
0	0	1
1	1	1

Examples:

1100 1101 = CD	1010 0001 = A1	1011 1000 = B8
0000 1111 = 0F	1001 1000 = 98	1111 0001 = F1
1100 1111 = CF	1011 1001 = B9	1111 1001 = F9

Truth Table
for XOR

	0	1
0	0	1
1	1	0

Examples:

1100 1101 = CD	1010 0001 = A1	1011 1000 = B8
0000 1111 = 0F	1001 1000 = 98	1111 0001 = F1
1100 0010 = 0D	0011 1001 = 39	0100 1001 = 49

INSTRUCTION TABLE

A:	ADD
AB:	ADD BYTES
ABS:	ABSOLUTE VALUE
AI:	ADD IMMEDIATE
ANDI:	AND IMMEDIATE
B:	BRANCH
BL:	BRANCH AND LINK
BLWP:	BRANCH AND LOAD WORKSPACE POINTER
C:	COMPARE WORDS
CB:	COMPARE BYTES
CI:	COMPARE IMMEDIATE
CLR:	CLEAR
COC:	COMPARE ONES CORRESPONDING
CZC:	COMPARE ZEROS CORRESPONDING
DEC:	DECREMENT
DECT:	DECREMENT BY TWO
DIV:	DIVIDE
INC:	INCREMENT
INCT:	INCREMENT BY TWO
INV:	INVERT
JEQ:	JUMP EQUAL
JGT:	JUMP ARITHMETIC GREATER THAN
JH:	JUMP LOGICAL HIGH
JHE:	JUMP HIGH EQUAL
JL:	JUMP LOGICAL LOW
JLE:	JUMP LOW EQUAL
JLT:	JUMP ARITHMETIC LESS THAN
JMP:	JUMP
JNC:	JUMP NO CARRY
JNE:	JUMP NOT EQUAL
JNO:	JUMP NO OVERFLOW
JOC:	JUMP ON CARRY
JOP:	JUMP ODD PARITY
LI:	LOAD IMMEDIATE
LWPI:	LOAD WORKSPACE POINTER IMMEDIATE
MOV:	MOVE A WORD
MOVB:	MOVE A BYTE
MPY:	MULTIPLY
NEG:	NEGATE
ORI:	OR IMMEDIATE
RTWP:	RETURN (WITH OLD) WORKSPACE POINTER
S:	SUBTRACT
SB:	SUBTRACT BYTES
SLA:	SHIFT LEFT ARITHMETIC
SOC:	SET ONES CORRESPONDING
SOCB:	SET ONES CORRESPONDING BYTE
SRA:	SHIFT RIGHT ARITHMETIC
SRC:	SHIFT RIGHT CIRCULAR
SRL:	SHIFT RIGHT LOGICAL
STST:	STORE STATUS
STWP:	STORE WORKSPACE POINTER

REFERENCE

EASY-BUG

".":	CANCEL A COMMAND
"M"	INSPECT AND/OR CHANGE CPU MEMORY
"V"	INSPECT AND/OR CHANGE VDP MEMORY
"E"	EXECUTE MACHINE LANGUAGE PROGRAM
"S"	SAVE CPU MEMORY
"L"	LOAD CPU MEMORY

LINE-BY-LINE

" AORG"	SPECIFY A VALUE TO THE ASSEMBLER LOCATION COUNTER
" BSS"	RESERVE A BLOCK OF MEMORY
" DATA"	INITIALIZE MEMORY
" EQU"	EQUATES A LABEL WITH A VALUE
" TEXT"	ENTER A STRING OF ASCII
" END"	EXIT ASSEMBLER

MIMI-MEM EQUATES

VSBW	>6024
VMBW	>6028
VSBR	>602C
VMBR	>6030

KSCAN	>6020
>8374	CONTAINS KEYBOARD DEVICE NUMBER
>8375	RETURNS ASCII VALUE OF KEY
>837C	GPL STATUS REGISTER

>8C02	VDPWA: VDP WRITE ADDRESS REGISTER
>8C00	VDPWD: VDP WRITE DATA REGISTER
>8800	VDPRD: VDP READ DATA REGISTER

GLOSSARY

>A: HEX DIGIT EQUAL TO 10 IN DECIMAL

ADDRESS: THE WAY TO IDENTIFY ONE OF 65535 POSSIBLE MEMORY LOCATIONS

AND: LOGICAL OPERATOR SIMILAR TO "*": 1 AND 1 = 1, 1 AND 0 = 0

>B: HEX DIGIT EQUAL TO 11 IN DECIMAL

BIT: BINARY DIGIT

BINARY: NUMBER SYSTEM BASE 2

BREAK POINT: USED FOR TRACING A PROGRAM

BYTE: TWO NIBBLES - EIGHT BITS - ONE HALF A WORD

>C: HEX DIGIT EQUAL TO 12 IN DECIMAL

CHAIN: A NUMBER OF LINKS

CONTEXT: ENVIRONMENT DEFINED BY A SET OF WORKSPACE REGISTERS.

CPU: CENTRAL PROCESSING UNIT

>D: HEX DIGIT EQUAL TO 13 IN DECIMAL

>E: HEX DIGIT EQUAL TO 14 IN DECIMAL

>F: HEX DIGIT EQUAL TO 15 IN DECIMAL

GPL: GROM PROGRAMMING LANGUAGE

GROM: GRAGHIC READ ONLY MEMORY. SEQUENTIAL IN NATURE

HEXADECIMAL: NUMBER SYSTEM BASE 16

HIGH BYTE: LEFT BYTE OF A WORD

INDIRECT: USE OF A REGISTER AS A POINTER

LINK: A WAY TO TIE TWO THINGS TOGETHER

LOW BYTE: RIGHT BYTE OF A WORD

NIBBLE: ONE HEXADECIMAL DIGIT - FOUR BITS LONG

OR: LOGICAL OPERATOR SIMILAR TO "+": 1 OR 1 = 1, 1 OR 0 = 1

PROGRAM COUNTER: A SYSTEM REGISTER THAT INDICATES THE ADDRESS OF THE NEXT INSTRUCTION

RAM: RANDOM ACCESS MEMORY

REGISTER: A WORD USED FOR A SPECIAL PURPOSE

STATUS REGISTER: A SYSTEM REGISTER THAT CONTAINS FLAGS THAT INDICATE THE STATE OF THE COMPUTER. SEE PAGE 40 ED/ASM.

VDP RAM: NOT REALLY RAM; ACTS LIKE SEQUENTIAL READ-WRITE MEMORY. USED BY VIDEO DISPLAY PROCESSOR & BASIC INTERPRETER INFORMATION IN VDP CANNOT BE EXECUTED DIRECTLY BY THE MICRO

PROCESSOR

WORD: TWO BYTES - 16 BITS

WORKSPACE POINTER: A SYSTEM REGISTER THAT INDICATES THE CURRENT ACTIVE SET OF WORKSPACE REGISTERS

WORKSPACE REGISTER: ONE OF A SET OF 16 REGISTERS

XOR: EXCLUSIVE OR - ONE OR THE OTHER BUT NOT BOTH

EVALUATION CARD

COMMENTS ON "TUTOR":

QUESTIONS YOU HAVE:

DID BITS YOU WOULD LIKE TO SHARE:

REQUESTS FOR FURTHER "TUTORS" AND/OR POSSIBLE NEWS LETTER

OPTIONAL:

NAME: ______________________________

ADRESS: ______________________________

____

THE SOFTIES

7300 GALLAGHER DR. #229

EDINA, MN. 55435